REAL-TIME SLIDING PHASE VOCODER USING A COMMODITY GPU

Russell Bradford, John ffitch, Richard Dobson

Department of Computer Science
University of Bath, UK
jpff@cs.bath.ac.uk

ABSTRACT

We describe a new approach to the processing of audio by way of transformations to and from the frequency domain. In previous papers we described the Sliding Discrete Fourier Transform (SDFT), comprising an extension to the classic phase vocoder algorithm to perform a frame update every sample. We proposed this as offering musical advantages over the common STDFT, including lower latency and the potential for new classes of effect. Its major disadvantage has been the very high computational cost, which makes it intractable for real-time use even on high-specification consumer workstations.

We report on a version of the SDFT that exploits the intrinsic parallelism of the scheme on a commodity GPU, to implement the transform and its inverse in real time for multiple audio channels. This implementation is used to provide previously uncomputable real-time effects. We describe a key new effect, enabled by the method, which we have called Transformational FM (TFM).

1. INTRODUCTION

There are many musical analyses and transformations that are best performed in the frequency domain. For example Lazzarini *et al.* [7] have introduced a range of spectral operations for Csound.

These transformations are predicated on an efficient translation from the time domain to frequency, and the inverse translation. For many years the usual process has been the FFT algorithm, with its much repeated $n\log(n)$ time complexity, coupled with overlapping windows. For a number of years this STDFT mechanism has been available in real time. However it was noted by some that as the overlap between the windows became larger the perceived quality improved.

In the limit if the windows overlap by all but one sample we consider the sliding form[1]. There is an alternative to the FFT/IFFT algorithm if the advance is one sample at a time. In [2] it is argued that there are a number of significant advantages of the single sample advance, but that this advantage is critically undermined by the greatly increased computational cost over the standard "hopping" phase vocoder. In this paper we show how with reasonably cheap commodity hardware we can deliver significantly more performance than is needed for real-time.

2. THE UNDERLYING MATHEMATICS

The Discrete Fourier Transform (DST) is defined by the formula

$$F_t(n) = \sum_{j=0}^{N-1} f_{j+t} e^{-2\pi i jn/N} \tag{1}$$

where the PCM-coded input signal is f_t, and $F_t(n)$ are the n frequency (complex) amplitudes for time t, and N is the (assumed) cyclic period of the signal.

If we know the values $F_t(n)$ we can determine $F_{t+1}(n)$:

$$F_{t+1}(n) = \sum_{k=0}^{N-1} f_{k+t+1} e^{-2\pi i k \frac{n}{N}} \tag{2}$$

$$= \sum_{k=1}^{N} f_{k+t} e^{-2\pi i (k-1) \frac{n}{N}} \tag{3}$$

$$= \left(\sum_{k=0}^{N-1} f_{k+t} e^{-2\pi i k \frac{n}{N}} - f_t + f_{t+N} \right) e^{2\pi i \frac{n}{N}} \tag{4}$$

$$= \left(F_t(n) - f_t + f_{t+N}\right) e^{2\pi i \frac{n}{N}} \tag{5}$$

Thus the cost of moving on by one sample is N complex multiplications and $2N$ additions, as the value of $e^{2\pi i \frac{n}{N}}$ can be precomputed.

There are two immediate problems that need to be addressed in this formulation. Firstly it is well-known that applying a window such as the Hamming window reduces spectral smear, and with this formulation the window cannot be applied in the time domain. The solution in this case is to apply the window as a frequency-domain convolution. That is to say, it can be applied after the SDFT as multiplication of the spectral transform of the window. Indeed for cosine-based windows this operation is simple (see [5]).

The second issue is the amount of accumulated error, as numerical rounding errors may propagate. These errors relate primarily to the accumulating phase advance (idiomatic to the phase vocoder), here computed per sample. Our experiments show that if the SDFT is performed in single precision IEEE arithmetic and at the CD sampling rate, phase errors at the resynthesis stage (which is in effect additive synthesis by oscillator bank) become evident as distortion of the waveform after as little as ten minutes. In contrast, using double precision we observe no deterioration after a day. We note here as a general point that whenever such processes (originally implemented as an "offline" process applied to a relatively short soundfile)

are recast as real-time effects, it is usually important to perform some form of "soak test" of this nature to confirm that errors of this kind do not accumulate to detrimental effect.

In the paper by Moorer ([8]) a complicated inverse formula is developed. However it requires N^2 data to be maintained and is clearly impractical, the more so for a real-time implementation. Instead we use a direct calculation of the definition of the inverse DFT:

$$f_t = \frac{1}{N} \sum_{n=0}^{N-1} F_t(n) e^{2\pi i t n/N} \quad (6)$$

but as we only need consider one value of t for each frame this is more efficient than the formula would suggest.

To deliver a useful SDFT we must be able to perform these operations, any spectral transformation operation, and an inverse within the constraints of real-time performance.

3. THE HARDWARE

In [2] attempts to perform the transformation and its inverse on standard hardware were shown to be much too slow for other than proof-of-concept, and the authors advocated the use of special plug-in vector hardware to give sufficient power; in particular the Clearspeed CSX board. There are indications that this hardware is viable [9], but it is specialised and relatively expensive hardware aimed primarily at the HPC community, and is not widely available.

In this paper we report on the use of a more common parallel computing component, the video card. Recent graphics cards incorporate significant amounts of parallel processing to support, for example, 3-D games. The release of the Nvidia CUDA language for some of their GPU-based cards opens the possibility of using this bulk parallelism for audio processing. We are using an Nvidia GeForce GTX 470, a medium-range graphics coprocessor. This card has has 14 *streaming multiprocessors* (SMs), each of which has 32 processing cores, giving a total of 448 cores. A general feature of such coprocessors is that each core in a given SM is has to run exactly the same code, in lockstep with every other core in that SM. In the context of a graphics processor, where we might want to run the same processing over many pixels, this is not a constraint: each core processes an individual pixel in parallel. For our purposes, this translates to the cores processing individual bins in the DFT in parallel.

The cores run at 0.6GHz, and support IEEE 64-bit (double) floating point. This is a relatively new feature for graphics cards; previous generations of video cards tended only to support 32-bit (single float) precision. As noted above, this is important to us (while it has been less important to graphics processing, hence its late arrival in such cards). The card has a theoretical performance of about 1000 GFlops.

As a comparison, the card resides in a standard 4-core Intel i7 desktop machine, running at 2.67Ghz, with a theoretical performance of approximately 40 GFlop. The relative slowness of the cores in the coprocessor is more than offset by their multiplicity. But we have to work hard to get at this large pool of computational potential.

The card has 1.3GB of its own memory, shared between all the SMs. However, and this turns out to be critical in the design of CUDA programs, access to that memory is desperately slow, taking 100s of clock cycles per access. To mitigate this, each core supports multithreading in hardware, trivially supporting 1000s of threads in a typical program. To hide the memory access cost, the hardware will rapidly swap between threads waiting for data and threads waiting for processing time. Having many threads means that there will always be some processing to do, rather than waiting idle on data access. An efficient CUDA program needs to be written with this in mind.

The card also contains a limited amount of fast shared memory (across the cores in an SM) , of the order of 10s of KB, together with some other more specialised kinds of memory. An efficient CUDA program will need to consider the use of these memory blocks, maybe copying data between slow global memory and fast local shared memory. This all adds complexity to the program.

This is above the usual issues with coprocessors that do not have direct access to the main system's memory. We must also consider the cost in time of moving data back and forth from the coprocessor. A program needs to balance the cost of data movement against the gains achieved from using the coprocessor. Audio presents special challenges in this respect - we typically wish to transfer as small a data packet as possible to minimise latency, and this can directly conflict with the optimal use of the parallel hardware.

We are not the first to use GPUs in audio processing. There have been occasional presentations in CUDA conferences relating to audio. [4] used a GPU for a variety of filtering operations, but only got improvements over a standard CPU in an implementation of reverberation. Indeed it is our experience that organising these algorithms for a GPU is a non-trivial operation.

4. THE SOFTWARE

Our demonstration software is written in a combination of C and CUDA. The C frontend obtains the input samples and feeds them to the DFT engine running on the GPU, together with control data, and retrieves the processed audio. There is also an interactive real-time demonstrator with a trivial GUI, written in C++, that allows us to control some simple FM transformations (applied in the frequency domain), including the special case where the modulation can be performed at audio rates. We describe the transformations later (section6).

As is clear from section3, the critical aspect of the program is the management of memory access. Our current code is the result of many experiments and arrangements, too numerous to catalogue. The code has been validated by checking that the resynthesised sound data is the same

as the input if no transformations are performed, and by checking a number of simple pitch-shifts.

5. PERFORMANCE

No of Channels	Time for 1sec sound
1	0.56 sec
2	0.59 sec
4	0.70 sec
8	0.88 sec

Table 1. Times for 1sec round trip multi-channel sound with 1024 bins.

We do not make any claims for optimality of our program, but it is sufficiently fast. We have been working at a sample rate of 44100 Hz, using double precision IEEE floating point arithmetic. For the first measurements the GPU program calculates the complex DFT by the sliding method, applies a Hamming window, and then converts the data into the magnitude-frequency form that is needed in the standard phase vocoder. In fact this is the most expensive operation, involving the calculation of an arctangent. The data is then converted back to the complex form, and resynthesised using equation (6). The size of the window was 1024 samples.

For a mono input stream this round trip runs at about twice real-time. However, doubling the number of channels does not double the time required; the times shown in table 1 show that the program is not saturating the available processing power. This is an effect of the memory access issues describe above, and suggests that there may be more musical use that can be performed with this commodity hardware.

6. APPLICATION

Our motivation for this work is not an exercise in computer science, but a desire for clean and efficient musical sonic transformations. To this end we have implemented two demonstration applications, both of which are well inside real-time.

The first is a straightforward pitch-shifter. As demonstrated in [2] we take advantage of the fact that the conversion of the raw SDFT frames into sliding phase vocoder frames of magnitude/frequency bins leads to a working bin bandwidth of $\pm$ Nyquist. Resynthesis does not employ the IDFT as such (which would impose the usual restrictions on bin frequencies) but uses direct additive synthesis, thus we can have any frequency in any bin. After the initial SDFT analysis stage we are free to view the process entirely in terms of a free-ranging oscillator bank. This means (disregarding any phase-locking considerations) that pitch-shifting is reduced to the trivial case of scaling each bin. This is controlled by a simple GUI controlled by the main processor. Via a slider the amount to shift the sound as a ratio (1 being no change, 2 an octave up and 0.5 an octave down) is passed continuously to the SDFT engine on the GPU. The input can be taken from an audio file or directly from a microphone. There is nothing exceptional about this program apart from the clarity of the transformation.

The second application is a variation of the first program with one significant difference. Instead of just a pitch shift, the shift of pitch is sinusoidal, with added GUI controls for the amplitude of sinusoid and the frequency of the change (figure 1). If the input were a pure tone then this would be just frequency modulation. The innovation here is that we now have a choice of modulation strategy. We can, following the standard formulation for audio-rate FM, modulate the input sound (such as a monophonic instrumental voice) as a whole, or we can analyse the incoming sound into its component frequencies, and then each component can be individually subjected to the FM transformation; in each case the modulation can be applied (thanks to the single-sample update) at audio rates. This is what we define as Transformational FM (TFM), which was introduced in [2], but there was a slow off-line calculation. With the CUDA/CPU implementation this is now available as a real-time multi-channel effect processor. In figure 2 we show a screen shot of the application, with the waveform and spectrum of the input signal (top right) and output (bottom right).

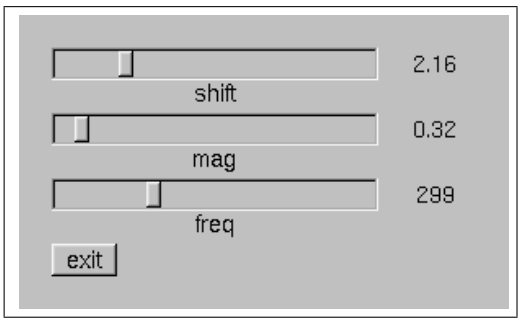

Figure 1. Screen shot of TFM GUI.

It is our contention that this effect is new, and now available at low cost to add to the existing repertoire of real-time effects.

7. CONCLUSIONS AND FUTURE WORK

We believe that the next big challenge for audio processing is to harness parallel processing in a musical way. We describe this approach High-Performance Audio Computing (HiPAC). Unlike standard High Performance Computing (HPC), HiPAC demands that we be especially aware of latency and the many demands of real-time operation. This has already been expounded in [3] and [6], and discussed in a panel at ICMC in 2008 [10].

The current program is not fully optimised, but can already handle 8 channels. As presently implemented, we modulate the incoming audio by a simple fixed frequency. This gives a result loosely comparable to ring modulation.

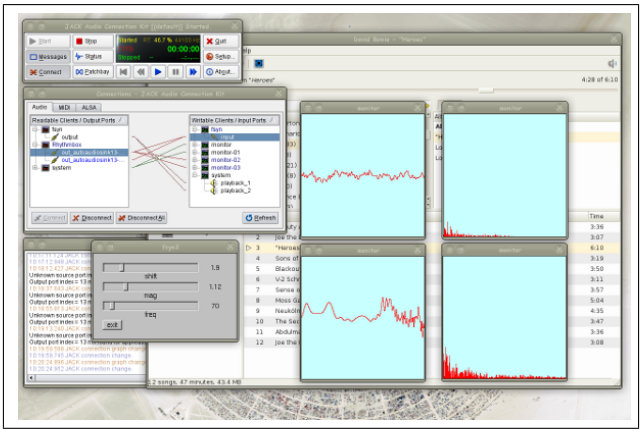

Figure 2. Screen shot of TFM in action.

A next stage is to incorporate a simple pitch tracking element, which will enable the FM Modulation Index to be preserved as the source changes pitch. A further obvious extension of the program would be to implement the constant-Q variant of the SDFT [6]. This requires three SDFTs for each frequency bin, so as we can perform the basic operations on 8 streams at present it seems likely that at least stereo constant-Q is realisable, assuming the all-important data positioning issues can be resolved.

The work presented here is a significant step towards the target of a broader application of HiPAC techniques. Not only is the DFT performed in real time, which has been the case for some years, but the frequency domain representation is updated at every sample, not only offering the consequent improvements in audio quality but also, and more significantly, allowing new families of transformations to become available to the performing musician. We note in particular the fact that the resynthesis step is performed by direct additive synthesis, a process that is well-known but also generally considered as computationally excessively demanding. We have highlighted TFM as one way of exploiting the single-sample streaming analysis of the sliding phase vocoder. In the general case, any process that can be applied to the output of a phase vocoder style oscillator bank can be employed. Even allowing for the need to optimise such processes to take maximum advantage of the GPU architecture, the speed gains we have obtained via commonly available low-cost hardware suggest that there is much scope for the implementation of "GPU streaming plugins" that can transform the raw additive output of the process in a multitude of ways, with the further alluring prospect of multi-channel and polyphonic operation.

8. REFERENCES

[1] R. Bradford, R. Dobson, and J. ffitch, "Sliding is Smoother than Jumping," in *ICMC 2005 free sound*, SuviSoft Oy Ltd, Tampere, Finland, Ed. Escola Superior de Música de Catalunya, 2005, pp. 287–290.

[2] ——, "The Sliding Phase Vocoder," in *Proceedings of the 2007 International Computer Music Conference*, S. O. Ltd, Ed., vol. II. ICMA and Re:New, August 2007, pp. 449–452, iSBN 0-9713192-5-1.

[3] R. Dobson, J. ffitch, and R. Bradford, "High Performance Audio Computing – A Position Paper," in *Proceedings of the 2008 ICMC*. SARC, Belfast: ICMA and SARC, 2008, pp. 213–216, iSBN 0-9713192-6-x.

[4] F. Fabritius, "Audio processing algorithms on the GPU," Master's thesis, Kgs. Lyngby, Technical University of Denmark, 2009.

[5] J. ffitch, R. Dobson, and R. Bradford, "Sliding DFT for Fun and Musical Profit," in *6th International Linux Audio Conference*, F. Barknecht and M. Rumori, Eds., LAC2008. Kunsthochscule für Medien Köln: Tribun EU, Gorkeho 41, Bruno 602 00, March 2008, pp. 118–124, iSBN 978-80-7399-362-7.

[6] ——, "The Imperative for High-Performance Audio Computing," in *Proceedings Linux Audio Conference*. Istituzione Casa della Musica, 2009, pp. 73–79.

[7] V. Lazzarini, J. Timoney, and T. Lysaght, "Spectral Signal Processing in Csound 5," 2006, pp. 392–395.

[8] J. A. Moorer, "Audio in the New Millennium," *J. Audio Eng. Soc.*, vol. 48, no. 5, pp. 490–498, May 2000.

[9] I. Tsimashenka, "Development, Implementation and Analysis of Real-Time Parallel Algorithm of Sliding Discrete Fourier Transform," M.Phil., Department of Computer Science, University of Bath, Dec 2010.

[10] D. Wessel, R. Dannenberg, Y. Orlarey, M. Puckette, P. V. Roy, and G. Wang, "Reinventing Audio and Music Computation for Many-Core Processors." SARC, Belfast: ICMA and SARC, 2008, iSBN 0-9713192-6-x.